Razvoj hibridnih web aplikacija korištenjem Ionic platforme

Juran, Andrija

Undergraduate thesis / Završni rad

2019

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Rijeka / Sveučilište u Rijeci**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:195:759613

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

Sveučilište u Rijeci
Fakultet informatike
i digitalnih tehnologija

Repository / Repozitorij:

Repository of the University of Rijeka, Faculty of Informatics and Digital Technologies - INFORI Repository

zir.nsk.hr

DIGITALNA KNJIŽNICA

Image not found or type unknown

Sveučilište u Rijeci – Odjel za informatiku

Jednopredmetna informatika

Andrija Juran

Razvoj hibridnih web aplikacija korištenjem Ionic platforme

Završni rad

Mentor: dr. sc., Igor Jugo

Rijeka, 30.8.2019

Zadatak:

Izrada mobilne web aplikacije korištenjem „Ionic“ platforme. Hibridne web aplikacije je naziv za kombinaciju web tehnologija i određenih platformi koje njihov kod prevode u kod aplikacija za mobilne operativne sustave „Android“ ili „IOS“. „Ionic“ je platforma jedna od vodećih takvih platformi. U radu je potrebno dati pregled Ionic platfome te izraditi i testirati oglednu hibridnu aplikaciju prema zadanim zahtjevima.

Sažetak:

Za izradu hibridne web aplikacije odabrao sam temu rezervacije i ponude smještaja. Korištene su tehnologije „Angular-Ionic“ s „Firebase“ bazom podataka. Usmjeravanje na stranice aplikacije implementiralo se pomoću „AngularRouter-a“, a glavni dijelovi aplikacije na koje je korisnik usmjeravan jesu: smještaji, rezervacije i ponude. Backend se realizirao pomoću „REST API“ koji nudi Firebase koji naknadno komunicira sa bazom podataka. Također, implementiran je čuvar koji osigurava da korisnici koji nisu autentificirani ne mogu pristupiti niti jednoj stranici aplikacije osim autentifikaciji. Svaki samostalni, funkcionalni dio aplikacije predstavlja jednu komponentu aplikacije koja u programskom kodu ima svoj direktorij s potrebnim datotekama, kako je i uobičajeno za svaku „Angular“ aplikaciju. Programski kod je pisan u objektno orijentiranom TypeScript-u, a u većini koda koristi se rxjs library koji radi s observable varijablama.

Summary:

For developing hybrid web application I chose a subject of offering and booking accommodation. The technologies I used are Angular and Ionic framework with Firebase as a Database. Application routing is implemented using Angular router, and the main parts of the application are accommodations, reservations, and offers. Communication with the backend was made using REST API which afterward communicates with the database. Also, there is an authentication guard which ensures that users who are not authenticated can not access any of the application's pages except the authentication page. Every individual, functional part of the application represents one component which has its own directory in the application's code, as it is common for every Angular application. Application code is written in the object-oriented TypeScript, and in the majority of the code RxJs library is being used, which handles observables.

Ključne riječi: Ionic platforma, Angular, Book&Go, navigacija, komponenta, modul, dekorator, datoteka, direktorij.

Table of Contents

1. UVOD 1
2. Korištene tehnologije 2
2.1 Ionic Framework 4 2
2.2 Angular 7 3
2.3 Node.js 4
2.4 Capacitor 4
2.5 Firebase 5
2.6 Upute za korištenje aplikacije Book&Go 6
3. Cjelokupan proces razvoja web aplikacije 7
3.1 Navigacija i usmjeravanje 7
3.2 Primanje korisničkog unosa 11
3.2.1 Template orijentiran pristup 11
3.2.2 Reactive forms pristup 11
3.2.3 Uređivanje ponude smještaja 12
3.2.4 Obrazac za rezervaciju smještaja 13
3.3 Upravljanje podacima 14
3.3.1 Dodavanje novih smještaja 14
3.3.2 Ikona učitavanja 15
3.3.3 Ažuriranje smještaja 16
3.3.4 Slobodni smještaji 16
3.3.5 Rezervacija smještaja 16
3.4 Pozadinski servis podatka (Backend) 18
3.4.1 Smještaji 18
3.4.2 Rezervacije 19
3.5 Dodavanje Google Maps-a 20
3.6 Pristupanje opcijama uređaja (kamera i lokacija) 22
3.6.1 Pristupanje korisničkoj lokaciji 22
3.6.2 Pristupanje kameri uređaja 22
3.7 Autentifikacija korisnika 24
3.7.1 Registracija i prijava 24
3.7.2 Upravljanje tokenom 25
3.8 Stiliziranje aplikacije 27
4. Zaključak 29
5. Literatura 30
6. Internetski izvori 31

1. UVOD

Hibridna web aplikacija koju sam izradio zove se Book&Go. Služi za rezervaciju ponuđenih smještaja pa tako i izradu ponuda vlastitih smještaja. Funkcionira na način da se svaki novi korisnik mora registrirati te nakon toga prijaviti u aplikaciju. Nakon toga ponuđen mu je jedinstven prikaz aplikacije koji odgovara njegovim prijašnjim rezervacijama i ponudama ukoliko postoje. Postoje mogućnosti pregleda ponuđenih smještaja, smještaja koji su slobodni i njihova rezervacija, izrada vlastite ponude smještaja te pregled svih svojih rezervacija i ponuda. Za izradu aplikacije u glavnom fokusu koristila se „Ionic“ i „Angular“ platforma. Također, koristili su se alati poput: Node.js, Capacitor, Android Studio, Visual Studio Code i Firebase. Kompletan programski kod je pisan u objektno orijentiranom TypeScriptu. Aplikacija je implementirana za korištenje na webu i Android uređajima.

2. Korištene tehnologije

U nastavku dati ću kratak pregled svake tehnologije koje sam koristio za izradu aplikacije.

2.1 Ionic Framework 4

„Ionic framework“ je open source UI alat za izgradnju visoko kvalitetnih mobilnih i računalnih aplikacija korištenjem web tehngologija kao što su „JavaScript“, „HTML“ i „CSS“. „Ionic“ je fokusiran na frontend korisničkom sučelju i interakcijom sa aplikacijom. Može se koristiti samostalno ili u skladu sa ostalim alatima kao što je „Angular“. Cilj je izgradnja i razvijanje aplikacija koje funkcioniraju na više različitih platformi, kao što su „Android“, „iOS“, računala te na webu kao progresivna web aplikacija. Sve je bazirano na istom kodu pa zato i izreka: „piši jednom, koristi svugdje“. Kao jednostavan i funkcionalan, „Ionic“ je dizajniran sa komponentama, tipografijom, interaktivnim paradigmama i baznom temom koje samostalno funkcioniraju na svim platformama. „Ionic“ je potpuno besplatan, izdan s „MIT“ dozvolom, što znači da se može koristiti u osobne ili komercijalne svrhe. Ionic dolazi sa vlastitim naredbenim sučeljem koje programerima nudi niz korisnih naredbi pri gradnji aplikacije. Iako je namijenjen za korištenje sa ostalim alatima, Ionic sada nudi i korištenje potpuno neovisno o ostalim alatima, ubacivanje se omogućuje pomoću „<script>„ oznake u web stranici. Ionic komponente izgrađene su sa web standardima korištenjem Javascripta, HTML-a i CSS-a. Iako su prethodno definirane i izgrađene, dizajnirane su da budu vrlo prilagodljive tako da svaka aplikacija komponentu učini svojom, a što onda aplikaciji dozvoljava da ima vlastiti izgled i dizajn. Dakle, „Ionic“ komponente mogu biti lako tematizirane da globalno mijenjaju izgled preko cijele aplikacije. Također, svaka se komponenta prilagođava platformi na kojoj je korištena, što znači da će aplikacija skinuta sa Apple's App Store-a dobiti iOS temu, a aplikacija skinuta sa Android Play Store-a dobiti će Material Design temu, kao i aplikacija koja bude pregledavana u računalnom pretraživaču. Unatoč tome što omogućava sučelje za različtim karticama aplikacije, Ionic podržava paralelnu navigaciju koja može biti ungiježđena. U svakom slučaju, u ovoj aplikaciji korištena je „Angular Router“ navigacija što je svakako praksa koju „Ionic“ preporuča. Što se tiče tematiziranja „Ionic“ dolazi s „CSS“ varijablama koje omogućavaju aplikaciji da izgleda privalčno, dok se drži svih web standarda. Također, dostupan je i set boja koji se može izmjeniti. „Ionic“ nudi integraciju raznih alata kao što su: „PayPal“, „SQLite“, „FireBase“, „GoogleMaps“ i ostali. Angular platformu koristimo paralelno s „Ionicom“ iz razloga što se ne želimo previše fokusirati na kompleksnu logiku koda, upravljanje stanjima i usmjeravanje na

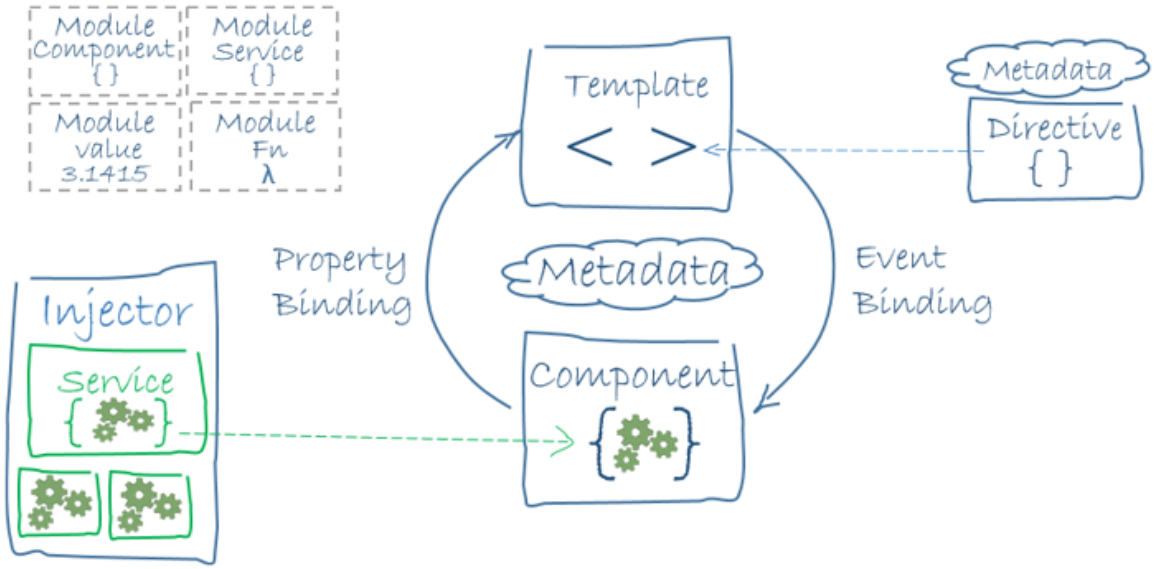
različite stranice aplikacije. Za korištenje „Ionica“ i „Angulara“ zajedno koristi se „@ionic/angular“ paket koji nam pruža Ionic komponente koje možemo korisititi na način kao da koristimo „Angular“ komponente moglo bi se reći. Dakle, u globalu kreira se „Angular“ projekt u koji se onda ubaci „Ionic“ modul, kako bi se komponente „Ionica“ mogle korisiti u običnoj „Angular“ aplikaciji.

2.2 Angular 7

„Angular“ je frontend framework za izgradnju aplikacija koristeći „HTML“ i „TypeScript“, a sastoji se od modula i komponenti. Temelji svake „Angular“ aplikacije su „NgModules“, koji skupljaju srodni kod u funkcionalne cijeline. Dakle, „Angular“ aplikacija je definirana cijelinom NgModula. Aplikacija uvijek ima korijenski modul koji omogućava početno podizanje, i također sadrži ostale module. Komponente definiraju tako zvane poglede, koji su cijeline zaslonskih elemenata između kojih Angular može birati i mijenjati, ovisno o programskoj logici i podacima. Komponente također koriste usluge (services), koje omogućavaju određene funkcionalnosti koje nisu direktno vezane s pogledima. Te usluge mogu biti dodane ostalim komponentama kao ovisnosti, a što čini programski kod više modularnim i efikasnijim. Komponente i usluge su u principu klase s dekoratorima koji označavaju njihov tip i pružaju meta podatke koji kazuju „Angularu“ kako ih treba koristiti. Meta podaci za klasu komponente povezuje je sa predloškom koji definira pogled. Pogled kombinira običan „HTML“ s „Angular“ direktivama i veznim oznakama koje omogućavaju Angularu da modificra „HTML“ prije nego ga renderira i prikaže na ekranu. Meta podaci za klasu usluge pružaju informacije koje „Angular“ treba omogućiti komponentama kroz dodavanje ovisnosti.

Znači u sažetku:

- komponenta i predložak zajedno definiraju Angular pogled.
  - Dekorator na komponenti klase dodaje meta podatke, uključujući pokazivač na povezani predložak
  - Direktive i vezivanje oznakama u predlošku komponente mijenjaju poglede ovisno o programskoj logici i podacima
- Uvozitelj ovisnosti pruža usluge komponentama, kao što su usluge usmjeravanja koje omogućavaju definiranje navigacije kroz aplikacijske poglede



Slika 1 - kako Angular funkcionira

2.3 Node.js

„Node.js“ je baziran na JavaScriptu, a uglavnom se koristi za razvijanje aplikacija na server strani koristeći „JavaScript“, iako ga se u ovoj aplikaciji nije koristilo na taj način. Prva stvar zbog koje se koristi je „Node Package Manager“ (npm), a koji se koristi za upravljanje zavisnostima kao što su knjižnice s treće strane. „NPM“ je defacto standarni upravitelj web paketima za web projekte generalno; dakle i za frontend aplikacije, čime se mi bavimo ovdje. „NPM“ se koristi za upravljanje zavisnotima, a „Ionic“ na neki način i je zavisnost moglo bi se reći. „Node.js“ je također potreban „Ionic CLI-u“ za korektno izvršavanje jer, u principu, alati koje „Ionic CLI“ vrti u pozadini bazirani su na „node.js-u“. Za to ne trebamo pisati nikakav kod ali moramo imat node instaliran kako bi ti alati normalno funkcionirali.

2.4 Capacitor

Capacitor je među-platformni „runtime“ koji olakšava razvijanje web aplikacija koje se izvršavaju na „iOS-u“, „Androidu“ i webu. Pruža konzistentu, web-fokusiranu cijelnu „API-a“ koji omogućavaju aplikaciji da ostane pri web standardima što je više moguće dok pristupa bogatim opcijama izvornog uređaja na platformama koje ih podržavaju. Dodavanje izvornih funkcionalnosti je jednostavno s „Plugin API“ za Swift na „iOS-u“, „Java“ na „Androidu“, „Javascript“ na webu. „Capacitor“ funkcionira na način da omota web aplikaciju u takozvan „WebView“ koji može prikazati web aplikacije unutar izvorne aplikacije. Dodaje most aplikaciji koja se izvršava u „WebView“-u, koji spaja programski kod od web aplikacije sa programskim kodom od izvorne aplikacije tako da dvoje mogu komunicirati. Na ovaj način, funkcionalnosti koje su inače dostupne samo izvornom kodu, mogu se koristiti u web aplikaciji,

i izvorni kod može komunicirati sa web aplikacijom koja se izvršava u „WebView“-u. Neovisno izvršava li se aplikacija na „iOS-u“, „Androidu“ ili na pretraživaču, uslužnik neke funkcionalnosti uvijek je isti za web aplikaciju i sva kompleksnost je zbrinuta od strane „Capacitora“ u pozadini.

2.5 Firebase

„Firebase“ je „Backend-as-as-Service (BaaS)“ platforma. „Firebase“ oslobađa developere od upravljanja serverima i stvaranja „API-a“. „Firebase“ je server, „API“, spremište podataka, napisan generalno kako bi se prilagodio mnogim potrebama. „Firebase“ je Real-Time baza podataka na koju se ne spaja kroz normalan „HTTP“, nego kroz web utičnicu, a koje su mnogo brže od „HTTP-a“. Ne moraju se izvršavati individualni pozivi web utičnica jer je jedna utičnica dovoljna. Svi podaci se automatski sinkroniziraju kroz tu utičnicu, koliko kod brzo klijentska mreža može podnijeti. Također, šalje nove podatke odmah kad su ažurirani. Firebase omogućuje spremanje podataka direktno sa klijenta na „Google Cloud Storage“. To spremište ima vlastita sigurnosna pravila za zaštitu od masa, dok dopušta potrebne privilegije autentificiranim klijentima. „Firebase“ nudi i email/password autentifikaciju koja se integrira direktno u bazu podataka tako da se koristi za kontrolu pristupa podacima. „Firebase“ također nudi i usluge poslužitelja za sve statične datoteke, koje poslužuje a globalnog „CDN-a“ s „HTTP/2“.

2.6 Upute za korištenje aplikacije Book&Go

URL: https://ionic-angular-fb9b3.firebaseapp.com

Klikom na url korisnik je usmjeren na prijavu u aplikaciju; ukoliko još nije izrađen račun klikom na gumb koji automatski prebaci na registraciju može se doći do obrasca za registraciju. Nakon toga korisnik stiže na početnu stranicu „svi smještaji“ u kartici „pronađi“ (prikaz kartica je na dnu), gdje se mogu vidjeti svi smještaji koje su ponudili svi korisnici. Klikom na „slobodni smještaji“ filtriraju se smještaji i korisniku su prikazani samo oni smještaji koje on smije rezervirati odnosno smještaji koje nije ponudio on nego ostali korisnici. Klikom na bilo koji od smještaja otvara se prikaz sa svim podacima o smještaju i ,također, gumb za rezervaciju koji pritiskom na njega otvara izbor rezervacije s manulanim odabirom datuma. Nakon toga korisnik može unijeti sve potrebne podatke za rezerviranje smještaja i klikom na gumb rezervacija se ostvaruje. Pri unosu podataka datuma korisnik je ograničen na datume koje je prijašnje izdavač smještaja odredio kao slobodni interval. Zatim, odlaskom na početnu stranicu i odabirom izbornika u gornjem lijevom kutu dolazimo do opcije „vaše rezervacije“ gdje korisnik može pogledati sve svoje rezervacije. Također, klizeći rezervaciju s desna na lijevo rezervacija se otkazuje.

Drugom dijelu aplikacije pristupa se putem odabira kartice „Ponudi“ gdje se izlistaju sve ponude vlastitih smještaja koje je korisnik kreirao ili ,ukoliko nema nikakvih ponuda, klikom na gumb „ponudi smještaj“ ili na ikonu „+“ , otvara se obrazac za kreaciju nove ponude. Pri kreaciji nove ponude korisnik unosi podatke koje želi. Pri unosu datuma određuje interval za koji je taj smještaj slobodan. Postoji mogućnost odabira lokacije putem samostalnog odabira ili autolociranja te ,također, direktno fotografiranje koje ukoliko se zatvori nudi odabir datoteka s uređaja. Pritiskom kvačice na desnome vrhu ponuda je kreirana i dodana u „Moje ponude“. Svaka ponuda može se uređivati klizeći po njoj s desna na lijevo. Za kraj korisnik se može odjaviti sa svog računa u izborniku u gornjem lijevom kutu aplikacije. Pri rezerviranju smještaja i kreiranju ponude korisnik će biti spriječen poslati obrazac ukoliko je završni datum razervacije/ponude manji od početnog datuma.

3. Cjelokupan proces razvoja web aplikacije

Prvi korak u izradi web aplikacije bio je instalacija svih potrebnih alata. Nakon toga, odabirom foldera i pokretanjem naredbe „Ionic start“, u cmd-u se stvara novi „Ionic-Angular“ projekt. Pri izradi projekta slijedilo je upoznavanje s „Angular“ platformom i njezinim komponentama, modulima, uslugama, te čitanje dokumentacije za „Ionic“ i „Angular“. Nadalje, slijedila je analiza i planiranje izgleda aplikacije, definiranje potrebnih funkcionalnosti, te u grubo skiciranje svake stranice u web aplikaciji.

3.1 Navigacija i usmjeravanje

Za upravljanje stranicama „Ionic“ koristi takozvani „Ionic StackController“ koji radi na sljedeći način: ako korisnik ide tipkom naprijed na slijedeću stranicu izvodi push page on stack, a ako korisnik ide tipkom natrag na prethodnu stranicu izvodi pop page of stack. Ionic cijeli taj stog stranica čuva u pomoćnoj memoriji.

Za početak izrade aplikacije krenuo sam s generiranjem svih potrebnih stranica koji će se nalaziti u aplikaciji pomoću naredbe „ionic generate page“, nakon čega se u projektu stvaraju gotovi direktoriji sa svim potrebnim datotekama za tu stranicu. Svaki direktorij na neki način reprezentira jedan funkcionalni dio apikacije(komponenta). Zatim slijedi definiranje usmjeravanja u web aplikaciji, u glavnom „app“ direktoriju stvara se nova datoteka „app.routing-module.ts“ gdje se pomoću tipa podataka „Routes“ koji nudi Angular definira usmjeravanje za tri dijela aplikacije:

1. Autentifikacija
2. Smještaji
3. Rezervacije

Autentifikacija je stranica na kojoj korisnik može birati između registracije i prijave i na temelju odabira ući u aplikaciju. Nadalje, „smještaji“ je skup stranica na kojima korsinik može nuditi i rezervirati smještaje. Na kraju, „rezervacije“ predstavlja stranicu koje prikazuje listu rezerviranih smještaja ovisno o svakom korisniku. Nadalje, u „places“ direktoriju stvara se „places-routing.module.ts“ datoteka u kojoj se ponovo pomoću Angularovom routera kreiraju usmjeravanja za skup stranica „smještaji“. U angularu usmjeravanje se implementira na način da se klikom na određeni gumb učita konkretno modul određenog direktorija u koji želimo navigirati te taj modul učitava potrebne komponente. Nadalje, za navigaciju kroz smještaje u aplikaciji, html datoteci komponente „smještaji“ („places-module.html“) kreira se pomoću

<ion-tabs> tagova, dvije kartice koje služe za prebacivanje između ponude i potražnje smještaja. U svim „routing“ modulima kroz aplikaciju koristio se tzv. „Lazy Loading“, koji funkcionira na način da se umjesto učitavanja i renderiranje cijele aplikacije korisniku učitaju samo potrebne sekcije koje je zatražio. Zatim slijedi još izrada dviju datoteka;

- Places-service – koja je zadužena za upravljanje podacima za sve smještaje. Ispred klase „PlacesService“ postavlja je dekorator „@Injectable“ što omogućuje umetanje u druge datoteke.
- Place-model – koja je prikazuje i stvara model smještaja

Slijedeće je izrada navigacije prema naprijed. Prvo se kreriraju proizvoljni smještaji na stranici „pronađi smještaje“. Nakon toga na svaki smještaj dodaje se navigacija koja vodi na stranicu „detalji smještaja“ ,pomoću „<ion-button>“ kojemu kao argument dodamo Angularov „[routerLink]“, na kojem onda odredimo stranicu koju želimo posjetiti pritiskom na gumb. Zatim na stranicu „detalji smještaja“ dodajemo „<ion-back-button>“ koji automatski prepoznaje koja stranica je prethodila trenutnoj i klikom na njega on nas vodi na tu stranicu. Međutim, ako osvježimo stranicu na kojoj se trenutno nalazimo „<ion-back-button>“ više ne zna koja stranica je bila prethodna pa mu treba dodati argument „defaultHref“ na koji će automatski navigirati ukoliko ne bude mogao identificirati prethodnu stranicu. Još jedna sitnica koju treba dodati u navigaciji je na stranici „Moje ponude“ gdje se dodaje „<ion-icon name=„add“>“ ikona koja vodi na stranicu izrade nove ponude. Sljedeća na red dolazi izrada navigacijskog izbornika koji se pali i gasi s lijeve strane aplikacije. U datoteci „app-component.html“ pomoću „<ion-menu>“ tagova unese se izbornik, a ikone u izborniku pomoću „<ion-item>“. Pri završetku izrade izbornika, slijedi ubacivanje u željene dijelove aplikacije. Dakle, u svakoj potrebnoj komponenti pristupimo html datoteci i u njoj „<ion-menu-button>“ unesemo gumb za otvaranje menija pri čemu se toj oznaci također može dodati id oznaka menija ukoliko ih imamo više.

```
src > app > places > offers > offers.page.html > ion-content > ion-grid > ion-row > ion-col
  1   <ion-header>
  2     <ion-toolbar>
  3       <ion-buttons slot="start">
  4         <ion-menu-button></ion-menu-button>
  5       </ion-buttons>
  6       <ion-title>Moje ponude</ion-title>
  7       <ion-buttons slot="primary">
  8         <ion-button routerLink="/places/tabs/offers/new">
  9           <ion-icon name="add" slot="icon-only"></ion-icon>
 10         </ion-button>
 11       </ion-buttons>
 12     </ion-toolbar>
 13   </ion-header>
 14
```

Slika 2 - Ubacivanje gumba za ion-menu u željenu datoteku

Kada radimo s izbornikom „Ionic“ napravi većinu posla za nas; konekretno se pobrine za izgled, dizajn te animacije, a omogućava nam jednostavno pozicioniranje izbornika na željenu stranicu. Jedino što nama preostaje je dodati elemente izbornika, odrediti usmjeravanje korisnika za klik na svaki od elemenata te dodati „<ion-menu-toggle>“ tagove oko svakoga elementa izbornika koji omogućuju automatsko zatvaranje menija nakon klika na bilo koji od njegovih elemenata. Zatim slijedi dodavanje još jedne komponente, „create booking“ u „booking“ direktoriju, koji će služiti kao modalni prikaz pri rezervaciji smještaja. Dakle, kada korisnik želi rezervirati smještaj otvoriti će mu se taj prikaz na kojemu će onda unositi podatke za rezervaciju. Taj modalni prikaz se implementira pomoću Angularovog „Modal-Controller-a“ u čiju se funkciju „create()“ proslijedi kompenenta aplikacije koja će biti prikazana modalno te podaci koji su potrebni toj komponenti.

```
app-routing.module.ts | place-detail.page.ts | create-booking.component.ts
src > app > places > discover > place-detail > place-detail.page.ts > PlaceDetailPage > constructor
117       });
118     }
119
120     openBookingModal(mode: 'select' | 'random') {
121       console.log(mode);
122       this.modalCtrl
123         .create({                                   KREIRANJE
124           component: CreateBookingComponent,        ULAZNI PODACI
125           componentProps: { selectedPlace: this.place, selectedMode: mode }
126         })
127         .then(modalEl => {
128           modalEl.present();                        PRIKAZ NA EKRANU
129           return modalEl.onDidDismiss();
130         })
131         .then(resultData => {
132           if (resultData.role === 'confirm') {      POVRATNI PODACI
133             this.loadingCtrl
134               .create({ message: 'Rezervacija smještaja...' })
135               .then(loadingEl => {
136                 loadingEl.present();
137                 const data = resultData.data.bookingData;
138                 this.bookingService
```

Slika 3- učitavanje modalnog prikaza

Zatim se u komponenti „create-booking“ pomoću „@Input“ dekoratora stvori varijabla (slika 4) koja će poprimiti vrijednost kojoj smo joj dodijelili u dijelu sa ulaznim podacima (Slika 3).

```
export class CreateBookingComponent implements OnInit {
  @Input() selectedPlace: Place;
  @Input() selectedMode: 'select' | 'random';
  @ViewChild('f') form: NgForm;
  startDate: string;
  endDate: string;
```

Slika 4 - deklaracija varijabla u komponenti modalnog prikaza

Nakon toga se ta varijabla smije korsiti kroz cijelu komponentu pa tako i u html prikazu kao što vidimo na slici 5.

```
<ion-header>
  <ion-toolbar>
    <ion-title>{{ selectedPlace.title }}</ion-title>
    <ion-buttons slot="primary">
      <ion-button (click)="onCancel()">
        <ion-icon name="close"></ion-icon>
      </ion-button>
    </ion-buttons>
  </ion-toolbar>
</ion-header>
```

Slika 5 - korištenje varijable u html-u

Na kraju, u zelenom okrivru slike 3. vidimo povrat podataka iz komponente(modalnog prikaza) koji onda služe pri daljnjem pozivanju funkcija za implmentaciju rezervacije smještaja. Kroz zadnje tri slike lijepo je prezentirano klasično slanje podataka kroz različite komponente koje se koriste u svakoj Angular aplikaciji, pa tako i u ovoj kao i mnogo slučajeva koji su slični ovomu.

Zadnji dio u ovoj sekciji aplikacije je dodavanje „auth guard-a“. On služi za sprečavanje pristupa korisniku u određeni dio aplikacije. U ovoj aplikaciji konkretno „auth-guard“ provjerava je li trenutni korisnik logiran u aplikaciju. Ukoliko je korisnik logiran dozvoljava ulazak na ostale dijelove aplikacije, a ukoliko nije izvodi usmjeravanje na autentifikaciju. Klasa „auth-guarda“ ima dekorator @Injectable, s toga se ubacuje u „routing-module“ i dodaje na potrebna usmjeravanja pomoću „CanLoad“.

3.2 Primanje korisničkog unosa

U ovom dijelu ćemo se najviše baviti koriničkim unosom putem obrazaca. Prvo je na redu planiranje na kojim dijelovima aplikacije će nam trebati obrasci i korinički unos. U aplikacijskom kodu koristio sam dvije vrste obrazaca:

- Reactive forms pristup
- Template orijentian pristup

3.2.1 Template orijentiran pristup

Za „Template driven“ pristup u korijenskom direktoriju u modul datoteci uvezemo „FormsModule“. Za unos u obrascu možemo koristiti „<ion-input>“, kao argument dodajemo „ngModel“. Argumente koje još možemo dodati su „required“ koji osigurava da se to polje ne smije ostaviti prazno i „minLength“ za osiguravanje minimalne dužine unosa u polje. Sve te unose omotamo s „<form>“ tagom u kojem postavimo proizvoljnu referencu „(f)“ na „ngForm“ vrijednost. Kasnije tu referencu smijemo koristiti gdje god želimo referencirati taj određeni obrazac. Za slanje obrasca korsitimo „(ngSubmit)“ kojeg postavimo na vrijednost funkcije „onSubmit()“ kojoj je argument referenca tog obrasca. Zatim se definira funkcija „onSubmit()“ u „component.ts“ datoteci komponente za koju radimo obrazac. Kao što vidimo pomoću argumenata „required“ i „minLength“ Ionic automatski određuje validaciju forme, i na „<ion-item>-e“ koji nistu validni poziva klase kao što su „ion-invalid“ i „ion-touched“. Nadalje, pomoću tih klasa možemo aplikaciji dodati nove funkcionalnosti ukoliko korisnik ne izvrši pravilan unos. Konkretno, u ovoj aplikaciji pri netočnom unosu podatka pri autentifikaciji, tekst u poljima se zacrveni i iskoči mala poruka koja opisuje problem. Skočna poruka se implementira pomoću “*ngIf“ koji se dodaje u tagu skočne poruke, a postavlja se na istinitu vrijednost ukoliko prethodno polje nije validno, a korisnik je već pokušao napraviti unos. Također na autentifikacijskoj stranici postavljen je gumb koji poziva funkciju „OnSwitchAuthMode()“ koja provjerava ako je korisnik u načinu prijave te ako je varijablu „isLogin“ postavlja na false čime je korisnik prebačen na registraciju i obrnuto.

3.2.2 Reactive forms pristup

Reactive forms pristup korišten je kod obrasca nove ponude odnosno pri korisničkoj izradi novog smještaja. Prvu u html datoteci korišten je „<ion-grid>“ sa „<ion-row>“ i „<ion-col>“ koji na jednostavan način strukturiraju prikaz podataka na stranici. Za reactive forms pristup prvo se u modul datoteci uveze „ReactiveFroms“, zatim se u typescript datoteci uveze „FormGroup“ i stvori atribut kao taj tip podataka. Forma se kreira u „ngOnInit()“ metodi koja

se prva pokreće pri inicijalizaciji, a pri kreaciji elemenata forme koristi se „FormControl“ koji se također uveze iz „@angular/forms“. Objektu „FormControl“ se proslijedi:

- inicijalna vrijednost
- updateOn – kada će se ažurirati unos
- validators – validacije koje element forme mora zadovoljiti

Za formu se kreira jedan objekt „FormGroup“, a za svaki element forme kreira se jedan objekt klase „FormControl“. Za sinkronizaciju forme odnosno povezivanje podataka iz html datoteke s podacima u typescript datoteci korisitimo direktive koje omogućava „ReactiveForms“. U html datoteci u „<form>“ tagu dodajemo argument „[formGroup]“ koji poprima vrijednost imena atributa obrasca u typescript datoteci, u ovom slučaju to ime je „form“. Nadalje, svakom elementu obrasca u html datoteci dodajemo argument „formControlName“ koji postavimo na vrijednost objekta kreiranih u typescript (.ts) datoteci. Vidi slike 5 i 6.

```
17
18  <ion-content>
19    <form [formGroup]="form">
20      <ion-grid>
21        <ion-row>
22          <ion-col size-sm="6" offset-sm="3">
23            <ion-item>
24              <ion-label position="floating">Title</ion-label>
25              <ion-input
26                type="text"
27                autocomplete
28                autocorrect
29                formControlName="title"
30              ></ion-input>
31            </ion-item>
32          </ion-col>
33        </ion-row>
```

Slika 6 – html datoteka izrade nove ponude

```
35  })
36  export class NewOfferPage implements OnInit {
37    form: FormGroup;
38
39    constructor(
40      private placesService: PlacesService,
41      private router: Router,
42      private loadingCtrl: LoadingController
43    ) {}
44
45    ngOnInit() {
46      this.form = new FormGroup({
47        title: new FormControl(null, {
48          updateOn: 'blur',
49          validators: [Validators.required]
50        }),
51        description: new FormControl(null, {
52          updateOn: 'blur',
53          validators: [Validators.required, Validators.maxLength(180)]
54        }),
55        price: new FormControl(null, {
```

Slika 7 – typescript datoteka izrade nove ponude

Zatim, na gumb za slanje obrasca koji se nalazi u gornjem desnom kutu stranice postavlja se „[disabled]“ vrijednost koja onemogućava slanje forme ukoliko forma nije validna.

3.2.3 Uređivanje ponude smještaja

Za izradu ovog obrasca ponovo se koristio ReactiveForms. Jedina razlika je u tome što sada pri izradi objekta elementa forme moramo proslijediti inicijalnu vrijednost polja.

3.2.4 Obrazac za rezervaciju smještaja

Za izradu ovog obrasca koristio se „Template“ pristup izradi. Pri izradi polja za unos broja osoba korisili su se „<ion-select>“ i „<ion-select-option>“ koji automatski korisniku nude padajući izbornik. S obzirom na to da pri izradi ponude smještaja postoji odabir datuma od kada do kada je moguće rezervirati taj smještaj, te podatke sada treba dinamički implementirati u ovome obrascu. Jedan problem koji se pojavio pri izradi ovog obrasca je da korisnci smiju unijeti datume na način da je početni datum rezervacije veći od završnog datuma rezervacije, što je bilo nužno spriječiti. To je izvedeno pomoću „@ViewChild“ dekoratora, koji prima ime reference forme, a dodan je u .ts datoteci u direktoriju za izradu rezervacije. On omogućava pristup unesenim podacima u formi koju referenciramo. U istu datoteku je zatim dodana funkcija „datesValid()“ koja vraća istinitu vrijednost ukoliko je završni datum veći od početnog datuma. Zatim se provjerava istinitost te funkcije u html datoteci. Unesenim podacima obrasca pristupa se u „onBookPlace()“ funkciji u .ts datoteci i spremaju se u Javascript objekt „bookingData“.

```
onBookPlace() {
  if (!this.form.valid || !this.datesValid) {
    return;
  }

  this.modalCtrl.dismiss(
    {
      bookingData: {
        firstName: this.form.value['first-name'],
        lastName: this.form.value['last-name'],
        guestNumber: +this.form.value['guest-number'],
        startDate: new Date(this.form.value['date-from']),
        endDate: new Date(this.form.value['date-to'])
      }
    },
    'confirm'
  );
}

datesValid() {
  const startDate = new Date(this.form.value['date-from']);
  const endDate = new Date(this.form.value['date-to']);
  return endDate > startDate;
}
```

Slika 8 - .ts datoteka za izradu rezervacije

3.3 Upravljanje podacima

U ovom dijelu bavimo se podacima koje korisnici unesu u aplikaciju, njima upravljamo u komponentama, zatim njihovo prosljeđivanje u tzv. „services“ koji se sastoje od raznih lista podataka, u ovom slučaju smještaja, rezervacija i korisnika. I još nam je potreban backend u koji se pohranjuju svi podaci kako se ne bi pri korisnikovoj odjavi ili deinstalaciji aplikacije izgubili.

3.3.1 Dodavanje novih smještaja

Datoteka s kojom ćemo u ovom dijelu puno raditi zove se „places.service.ts“. U njoj se bavimo podacima vezanim uz smještaje, a glavne metode koje sadrži su:

- dohvaćanje smještaja (getPlace)
- dodavanje smještaja (addPlace)
- ažuriranje smještaja (updatePlace)

Prvo uređujemo već izrađenu datoteku place-model koja nam služi kao baza odnosno prikaz kako bi smještaj trebao izgledati. Sastoji se od:

1. id
2. naslov
3. kratki opis
4. slika
5. cijena
6. slobodno od
7. slobodno do
8. userId

Svi ti podaci potrebni su nam kako bismo korektno upravljali smještajima, posebno „userId“ kako bismo znali koji korisnik je ponudio koji smještaj i na taj način spriječili korisnika da ne može rezervirati svoj smještaj. Počinjemo sa metodom „addPlace()“ u datoteci „places.service.ts“ koja prima sve podatke potrebne za izradu nove ponude. Tu metodu ćemo pozivati u metodi “onCreateOffer()“ koja se nalazi u datoteci „new-offer.ts“ u direktoriju za izradu nove ponude. Problem na koji se nailazi je ukoliko posjetimo stranice „potraži smještaj“ i „moje ponude“ i nakon toga pokušamo dodati novi smještaj, smještaj neće biti dodan jer se on trenutno dodaje u metodi „ngOnInit()“ koja se pokreće samo pri prvom otvaranju stranice. Rješenje za taj problem je korištenje „Subjecta“ iz „rxjs“ paketa. „Subject“ funkcionira na način da neki događaj ispuštamo u nekoj u komponenti, a zatim se u drugoj komponenti pretplatimo

na taj događaj. Na taj način svaki put kad dodamo novi smještaj „Subject“ će detektirati to novo dodavanje i osvježit će listu smještaja sa novim smještajem, te će svi smještaji na stranici biti korektno učitani. S toga, od sada pa nadalje, kada vraćamo listu smještaja u „getterima“ vraćamo je kao observable kako bismo se mogli pretplatiti na tu listu i onda dobiti njezin rezultat. Ukoliko želimo vratiti samo jedan element te liste možemo kombinirati funkciju „find()“ s operatorom „map()“ koji se implementira na svakoj vrijednosti koju je vratio observable i ispušta svaku vrijednost natrag kao observable. Kada se želimo pretplatiti na observable kako bismo došli do vrijednosti, moramo deklarirati „Subcription“ atribut u koji će se spremati vrijednosti kao observable, i nakon korištenja moramo u metodi „ngOnDestroy()“ taj atribut očistiti pomoću „unsubscribe()“ kako bi se oslobodila memorija. U suprotnome možemo naići na probleme kao što je curenje memorije.

```
export class PlacesService {
  private _places = new BehaviorSubject<Place[]>([]);

  get places() {
    return this._places.asObservable();  //objekt na koji se možemo pretplatiti
  }
```

Slika 9 - places.service.ts - ovdje vidimo stvaranje novog Subjecta i njegov getter sa observable

```
export class OffersPage implements OnInit, OnDestroy {
  offers: Place[];
  relevantPlaces: Place[];
  isLoading = false;
  private placesSub: Subscription;

  constructor(private placesService: PlacesService, private router:

  ngOnInit() {
    this.placesSub = this.placesService.places.subscribe(places =>
      this.offers = places;
    });

  ngOnDestroy() {
    if (this.placesSub) {
      this.placesSub.unsubscribe();
    }
  }
```

Dohvaćanje smještaja pomoću observable u ngOnInit i osobađanje memorije u ngOnDestroy

Slika 10 - new-offer.component.ts

3.3.2 Ikona učitavanja

Za korektno korisničko iskustvo ikona učitavanja je esencijalna. U kojoj god komponenti je želimo koristiti trebamo uvesti „LoadingController“ sa „@ionic@angular“. Implementira se pomoću „loadingController.create()“ i zatim prikaže na ekranu pomoću „present()“ i eliminira sa „dismiss()“, a možemo i dodati „setTimeout()“ kako bi cijela stvar

izgledala bolje. „setTimeout“ dodajemo u operator „map()“ koji se koristi u skladu sa observable, služi za izvođenje neke operacije koja neće utjecati na „observable“.

3.3.3 Ažuriranje smještaja

U „places.service.ts“ dodajemo metodu „updateOffer()“ koja prima id smještaja, naslov, cijenu, kratki opis i datume. U toj metodi uspoređujemo dobiveni id sa svim mjestima koje imamo na raspolaganju i tako lociramo smještaj koji želimo ažurirati. Nakon toga, na locirano mjesto stvorimo potpuno smještaj i proslijedimo mu sve stare podatke koje smo međuvremeno pohranili u pomoćnu varijablu; osim naslova i opisa koje mijenjamo a oni su već proslijeđeni kao argumenti toj metodi.

3.3.4 Slobodni smještaji

„Slobodni smještaji“ je druga stranica u segmentu pronađi smještaje, a sastoji se od smještaja koje korisnik smije rezervirati odnosno smještaja koje on nije ponudio. To je implementirano na način da je u html datoteku od pronađi smještaje dodan „<ion-segment“ s „<ion-segment-button>“. Tto nam omogućava da ovisno o tome je li korisnik odabrao sve smještaje ili slobodno smještaje, pošaljemo točnu informaciju u .ts datoteku. Po primitku te informacije u „onFilterUpdate()“ metodi, ona vraća sva sve smještaje (ako je korisnik odabrao sve smještaje) ili (ako je korisnik odabrao slobodne smještaje) obavlja uspoređivanje i pronalazi smještaje koji nisu od korisnika sa tom ID oznakom i onda vraća samo taj rezultat.

3.3.5 Rezervacija smještaja

U ovom dijelu ćemo raditi u „booking.service.ts“ datoteci koja se bavi podacima rezervacija. Ponovno ćemo napraviti „BehaviourSubject“ kao i za mjesta prethodno, ali ovoga puta će tip podataka biti „Booking“. Metode koje ćemo definirati su „addBooking()“ i „cancelBooking()“. Model tipa podataka „Booking“ će sadržavati sljedeće atribute:

- id rezervacije
- id smještaja
- id korisnika
- naslov smještaja
- slika smještaja
- ime
- prezime
- broj osoba

- rezervirano od
- rezervirano do¸
- ukupna cijena

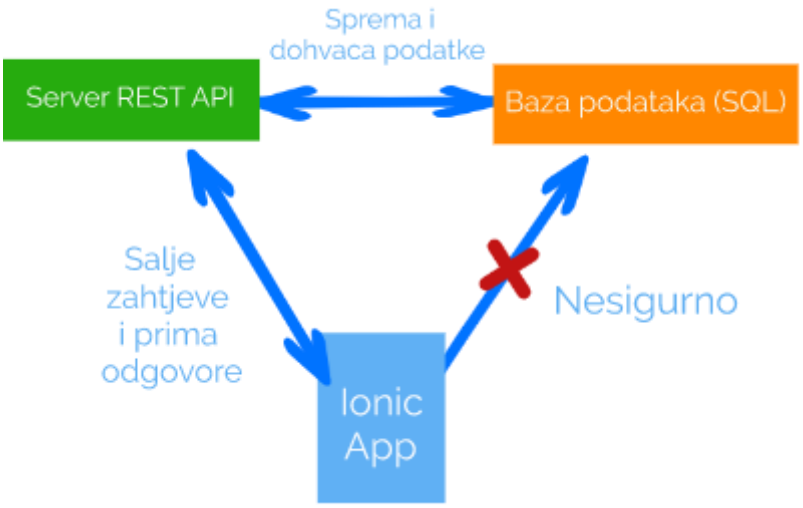
Nove rezervacije stvaramo u „addBooking()“ metodi:

- this.bookings.pipe(
  take(1), delay(1000),
  tap(bookings => this._bookings.next(bookings.concat(newBooking))))

U ovomu dijelu koda s pipe spajamo tri funkcije: u prvoj „take(1)“ pristupamo samo zadnje ažuirarnim rezervacijama, s „delay(1000)“ stvaramo odgodu od jedne sekunde te zatim s „tap()“ pristupamo rezervacijama koje smo dobili i dodajemo novu rezervaciju. Dakle, „addBooking()“ metodu ćemo pozivati u „place-detail.ts“ datoteci u metodi „openBookingModal()“ koja zapravo stvara modalni prikaz, dozvoljava unos u obrazac, i izvlači podatke iz obrasca koji se onda šalju u „addBooking()“ metodu. Iz te metode ćemo naknadno implementirati prijenos podataka na backend server. Za otkazivanje rezervacija metodi „cancelBooking()“ ćemo proslijediti id rezervacije koje želimo otkazati, zatim ćemo u metodi vršiti pretraživanje po svim rezervacijama uspoređujući svaki ID u funkciji „filter()“, koja će onda zadržati sve smještaje koji imaju različit ID od onoga koji smo proslijedili. Još jedna stvar koju trebamo napraviti je prikaz pomoćne poruke na stranici „Vaše rezervacije“; ukoliko ne postoji niti jedna rezervacija. To ćemo napraviti jednostavno u html datoteci komponente rezervacija, gdje ćemo provjeriti postoje li dostupne rezervacije te ako ne postoje ostaviti ćemo poruku. Zadnja stvar koju treba implementirati je ta da, ukoliko korisnik pristupi stranici sa smještajem koji ne smije rezervirati, gumb za rezervaciju mora biti nedostupan. To ćemo napraviti u „place-detail.ts“ datoteci u „ngOnInit()“ metodi gdje ćemo atribut „isBookable“ koji je inicijalno postavljen na pozitivnu vrijednost postaviti na negativnu vrijednost ukoliko se id korisnika poklapa sa id korisnika konkretno u smještaju. Zatim ćemo vrijednost tog atributa provjeriti u html datoteci koja će ukoliko je atribut negativan prikazati gumb za rezervaciju smještaja, koji onda nadalje otvara modalni prikaz obrasca za rezervaciju (create-booking.ts).

3.4 Pozadinski servis podatka (Backend)

U ovom dijelu baviti ćemo se prijenosom i dohvaćanjem podataka s vanjskog servera. Međutim što se Ionic aplikacije ona se ne spaja direktno na SQL server zbog sigurnosnih razloga. Umjesto toga spaja se na „REST API“ server koji možemo samostalno napraviti ili možemo korisiti uslugu kao što je Firebase. Pri izradi ove aplikacije koristio sam Firebase, koji nudi API sa kojim možemo komunicirati i onda naknadno taj API samostalno komunicira sa bazom podataka.



Slika 11 - Komunikacija sa serverom

Za komunikaciju sa serverom potrebno je u aplikacijski kod uvesti „HttpClientModule“ i „HttpClient“, pomoću njihovih servisa ostvaruje je jdosnotavna komunikacija sa serverom.

3.4.1 Smještaji

Za slanje podataka na Firebase koristimo „post()“ metodu koja prima dva argumenata:

1. url foldera na serveru
2. podatke koje želimo poslati u json obliku

Sada kada šaljemo podatke na Firebase on će automatski generirati jedinstveni id za svaku listu podataka koju mu pošaljemo. Upravo zbog toga u „post()“ funkciji na mjesto drugog argumenta ćemo također dodati „id: null“ oznaku i na taj način ćemo id ostaviti prazan kako bi Firebase samostalno to popunio. Pri dohvaćanju podataka (npr. fetchPlaces() u places.service.ts datoteci) koristimo „get()“ funkciju prije koje ćemo definirati tip podataka koji ćemo dohvaćati. Ponovo ćemo koristiti url direktorija kojem pristupamo na serveru, ali ovaj put nećemo proslijeđivati nikakve podatke. Pri definiranju tipa podataka koje ćemo dohvaćati morati ćemo definirati novi interface podataka, budući da ćemo u prvu ruku dohvaćati podatke bez id oznake kojoj ćemo

naknado pristupiti i dodati je. To radimo iz razloga što su na serveru podaci pohranjeni u „json“ obliku; dakle, za svaku id oznaku koja predstavlja jedan smještaj imamo određene podatke koje pripadaju toj id oznaci. Zatim, stvaramo pomoćno polje čiji će tip podataka biti onaj originalan sa id oznakom, tada u petlji na svaki element toga polja dodajemo podatke za jedan smještaj, i na taj način sada imamo pomoćno polje u kojem je svaki element jedan smještaj. I za kraj to pomoćno polje smještaja proslijedimo atributu „_places“ koji je „BehaviourSubject“ istog tipa podataka kao i naše pomoćno polje. To ćemo učiniti pomoću funkcije „next()“ koja prebriše i proslijeđuje nove podatke BehaviourSubjectu. Sa metodom „get places()“ možemo pristupiti atributu „_places“, i zatim tu metodu pozivati u kojoj god komponenti želimo pristupiti svim smještajima, i na taj način jednostavno prikazati sve podatke u html datoteci komponente.

Za ažuriranje podataka na Firebaseu koristimo „put()“ http funkciju koja nam omogućava da resurse koje lociramo možemo jednostavno zamjeniti novim resursima koje proslijeđujemo. Za primjer ažuriranje smještaja u datoteci usluge smještaja (places.service.ts) koristiti ćemo metodu „updatePlace()“. Ona će sadržavati „put()“ funkciju, koja će kao prvi argument primati URL datoteke u json formatu, a drugi argument polje koje već sadržava izmjenjene podatke smještaja. URL-u ćemo morati konkatenirati ID smještaja kako bismo točno mogli locirati smještaj na serveru. Unatoč tome, treba prvo pristupiti podacima smještaja, izmjeniti ih s argumentima proslijeđenim funkciji „updatePlace()“, i tek nakon toga pozvati „put()“ funkciju. Nakon što smo, poslali podatke na server ažurirati ćemo i lokalnu listu smještaja. Na sličan način funkcionira i metoda „addPlace()“, kod koje ćemo koristiti http funkciju „post()“ umjesto „put()“. Zatim za pravilno dohvaćanje podataka u nekoj od komponenata trebamo pristupiti lokalnoj listi smještaja i također pozvati funkciju „fetchPlaces()“ za dohvaćanje podataka sa servera i ažuriranje lokalne liste smještaja. Također je potrebno definirati metodu „getPlace()“, iz razloga što imamo u aplikaciji komponente kao što su uređivanje smještaja i prikaz detalja smještaja. Pristupanje podacima svih smještaja prilikom prikaza detalja samo jednog smještaja bilo bi poprilično ne ekonomično.

3.4.2 Rezervacije

Za dohvaćanje i slanje podataka rezervacija koristi se sličan princip kao i kod smještaja. Razlika je u tome što se metoda ažuriranja mijenja s metodom brisanja rezervacije, i također, definira se metoda za dohvaćanje svih rezervacija filtracijom po ID-u, kako bismo dobili i prikazali rezervacije koje pripadaju konkretnom korisniku. Koristi se i drugi „interface“ koji se kao i sve ostale metode vezane za smještaje definira u datoteci „booking.service.ts“.

3.5 Dodavanje Google Maps-a

Prva stvar kod dodavanje Google Maps-a je bilo nabaviti „google JavaScript API key“ koji se ubaci u aplikaciju. Njega ćemo ubaciti u „environment.ts“ datoteku da bude uvijek dostupan. Pomoću tog ključa dobivamo „JavaScript SDK“ sa kojim možemo uređivati google maps prikaz po vlastitom nahođenju. U aplikaciji potrebno je dodati dvije nove komponente:

1. google maps prozor – prikaz terena pomoću kojega odabiremo željenu lokaciju
2. birač – element u obrascu pomoću kojega se otvara google maps prozor

Obje komponente možemo dodati u novo stvoreni dijeljeni folder. U tom dijeljenom folderu stvaramo „module.ts“ datoteku koju onda možemo koristiti za uvoz obje komponente u druge komponente gdje su potrebne. Za birač napravimo jednostavnu html datoteku sa ikonom na koji pristiskom se izvodi metoda „onPickLocation()“, nadalje ta metoda služi za otvaranje prozora sa mapom. Prozor će se stvoriti sa „modalCtrl.create()“ i onda prikazati sa „present()“. Birač, zatim, možemo prikazati u html datoteci (npr. datoteka za stvaranje nove ponude) s oznakom „<app-location-picker>“ budući da je to ime postavljeno u komponenti tog birača.

```
@Component({
  selector: 'app-location-picker',
  templateUrl: './location-picker.component.html',
  styleUrls: ['./location-picker.component.scss']
})
```

Slika 12 - location-picker.component.ts datoteka

U ts datoteku komponente prozora mape dodajemo metodu „getGoogleMaps()“ bez argumenata, ona nam vraća „Promise“, a služi nam za učitavanje google maps „SDK“. Tu metodu ćemo pozivati unutar „Lifecycle“ točke koja se naziva „AfterViewInit“ a učita se nakon što se učitaju svi view-ovi na stranici. Unatoč tome što prijašnja metoda vraća „Promise“, ukoliko se google mapa ne učita točno dobiti ćemo kao rezultat grešku sa preodređenom porukom. Kod pozivanja metode dodajemo geografske širine i dužine na koje će se locirati mapa prilikom otvaranja. Pomoću „ViewChild()“ ćemo referencirati mjesto u html datoteci ( ) u kojem ćemo renderirati mapu. Pomoću funkcije „addListener()“, ćemo slušati svaki klik miša, i u varijablu ćemo pospremiti vrijednosti širine i dužine koje su postavljene klikom miša. Nakon toga ćemo dodati „modalCtrl.dismiss“ za zatvaranje prozora i proslijeđivanje vrijednosti. Zatim nam je potrebna funkcija koja će iz odabrane lokacije generirati konkretnu adresu u pisanom obliku koju možemo prikazati, nazvati ćemo ju „**getAdress()**“. Za izradu te

funkcije dobro je referirati se na „Geocoding“ deokumentaciju koja opisuje taj proces. Nadalje, potrebno je definirati i metodu koja vraća sliku mape, koja nam služi kao pregled na lokaciju koju smo odabrali u mapi. Za to se korisiti „Google static map api“ dokumentacija pomoću koje se šalju zahtjevi sa geografskom širinom, dužinom i zoom vrijedosti, također se odredi i veličina slike koja će biti prikazana. Tu sliku ćemo prikazati u html datoteci komponenete birača lokacije, a vraćat će je metoda „**getMapImage()**“. Bilo je potrebno dodati i odgovarajući izgled, kako bi odabrana slika bila korektno prikaza u obrascu. Dakle, u prikazu je ta slika zamjenila gumb „odaberi lokaciju“ u obrascu, a na nju je potrebno dodati funkcijonalnost da se pritiskom ponovo može birati lokacija kao i pritiskom na gumb prvi put. To ćemo učiniti s „(click)“ i onda to izjednačiti s metodom za koju želimo da bude izvršena.

Dakle finalno kako će sve funkcionirati je:

1. Pritiskom gumba ili slike u obrascu izvršava se metoda „**onPickLocation()**“, a ona nudi listu akcija gdje možemo birati želimo li sami odabrati lokaciju ili automatsko detektiranje (implementirano u slijedećem poglavlju). Pritiskom na samostalni odabir lokacije izvršava se metoda „openMap()“.
2. „**openMap()**“ metoda otvara komponentu google maps prozora i pohranju u javascript objekt vrijednosti geografske širine i dužine ovisno o lokaciji koju je korisnik odabrao. Zatim se taj objekt proslijeđuje „createPlace()“ metodi.
3. „**createPlace()**“ metoda će dobiti geografsku širinu i dužinu i pomoću tih podataka doći do adrese u string formatu sa metodom „**getAdress()**“, i također do slike lokacije sa metodom „**getMapImage()**“.

Zadnja stvar u „createPlace()“ metodi je izvršavanje događaja (EventEmitter), i putem njega, slanje podataka lokacije tamo gdje su potrebni. Konkretno u ts datoteci izrade nove ponude pristupamo podacima toga događaja pomoću „patchValue()“ šaljemo ih u obrazac. Zatim je potrebno još učitati mapu u komponentu detalji smještaja, tu stranicu korisnik otvara kad pregledava smještaj. Dakle, u ts datoteku trebamo dodati metodu koja će otvoriti prozor mape ali samo u opciji pregledavanja. Za to ćemo korisiti „selectable“ atribut koji ćemo ako se nalazimo u samo pregledavanju, postaviti na negativnu vrijednost, a s druge strane ćemo u „lifecylce“ metodi provjeravati vrijednost toga atributa i ovisno o njemu omogućiti ili onesposobiti biranje lokacije prilikom pregledavanja mape.

3.6 Pristupanje opcijama uređaja (kamera i lokacija)

U ovom poglavlju fokusirati ćemo se na korištenje opcija mobilnih i drugih uređaja. Koristi se Capacitor koji nam asistira pri korištenju tih opcija. Fokusirati ćemo se na korištenje korisnikove trenutne lokacije kao određivanje lokacije smještaja i korištenje kamere uređaja za slikanje pri dodavanju fotografije smještaju.

3.6.1 Pristupanje korisničkoj lokaciji

Za početak treba uvesti capacitor plugin u modulu korijenskog (aplikacijskog) direktorija. U ts datoteci komponente birača lokacije, stvorit ćemo metodu „**locateUser()**“ koja će nam služiti za lociranje trenutnog korisnika. Za to će nam biti potreban Geolocation plug-in. Za korištenje tog plug-ina treba postaviti razna dopuštenja u datotekama aplikacije, kako je navedeno u capacitor dokumentaciji. Prisupanje korisničkoj lokaciji pomoću Capacitor plug-ina je relativno jednostavno. U prethodno navedenoj metodi ćemo prvo provjeriti ako je plug-in dostupan te ako zbog nekog razloga nije prikazati poruku da se pojavila greška pri pristupanju lokaciji. Nadalje, ako je plug-in dostupan, s „Plugins.Geolocation.getCurrentPosition()“ stvorit ćemo „Promise“ koji će u „then()“ bloku sadržavati javascript objekt za spremanje koordinata i zatim taj objekt proslijediti metodi „**createPlace()**“, a čija je funkcija objašnjena u prijašnjem poglavlju. „Catch“ blok od „Promisa“ sadržavati će kod sa prikazvinje poruke o greški.

3.6.2 Pristupanje kameri uređaja

Također, i za kameru se treba referirati na dokumentaciju kako bismo postavili sve potrebne dozvole. Za početak treba kreirati komponentu birača slike na mjestu „/app/shared/pickers/image-picker“. Html datoteka te komponente će izgledati slično kao i kod komponente birača lokacije, tako da html kod možemo kopirati i djelomično ga urediti. U ts datoteci komponente birača slike dodati ćemo metodu „**onPickImage()**“.

U toj metodi ćemo kao i prije prvo provjeriti ako nam je plug-in dostupan, i zatim pozvati „Plugins.Camera.getPhoto()“ u kojoj ćemo postaviti razne atribute:

1. Quality: koji ne smijemo postaviti previsoko, to je praksa za sve web aplikacije i web stranice
2. Source: postaviti ćemo na prompt, što nam omogućuje da korisnik odabere želi li slikati sliku ili odabrati sliku iz galerije uređaja. Inače se može još postaviti na Photo i Camera, te se tako omogući samo jedna opcija
3. correctOrientation: za korektni prikaz pri horizonatlnom fotografiranju
4. width/height: za određivanje veličine slike
5. resultType: određujemo rezultat koji će nam se vratiti, ovdje je određeno Base64 string

Također, stvaramo „Promise“ i u „then()“ bloku pokrećemo događaj, koji pozivamo u html datoteci komponente izrade nove ponude, taj događaj poziva metodu „**onImagePicked()**“. Toj metodi koja se definira u ts datoteci komponente nove ponude, proslijeđuje se „base64 string“ koji smo odredili kao rezultat. U html datoteci nove ponude trebamo također dodati element u kojem će se nalaziti birač slike komponenta koju izrađujemo u ovom poglavlju. Tom elementu ćemo dodati isti css kod kao i kod birača lokacije. Treba obratiti pažnju i na uređaje koji nemaju kameru i za to kreirati odgovarajuću „fallback“. Za to ćemo i html datoteci birača slike kreirati html input za biranje slike iz galerije, a kojeg ćemo automatski kliknuti/odabrati ako je vrijabla „usePicker“ istinita. Tu varijablu ćemo postaviti na istinitu vrijednost ukoliko se korisnik nalazi na desktop računalu (slika 12).

```
if (
  (this.platform.is('mobile') && !this.platform.is('hybrid')) ||
  this.platform.is('desktop')  //ako je if true onda znaci da smo na desktopu
) {
  this.usePicker = true;
```

Slika 13 - ngOnInit u ts datoteci birača slike (image-picker.component.ts)

Zatim, u „onPickImage()“ u if grananju ako kamera plug-in nije dostupan, pomoću reference (kreirana sa „viewChild()“ i referira na html input element za odabit fotografije), ćemo simulirati klik mišem i prozor za odabir slike iz galerije će se otvoriti.

```
<input
  type="file"
  accept="image/jpeg"
  *ngIf="usePicker"
  #filePicker
  (change)="onFileChosen($event)"
/>
```

Slika 14 - html datoteka komponente birača slike

```
export class ImagePickerComponent implements OnInit {
  @ViewChild('filePicker') filePickerRef: ElementRef<HTMLInputElement>;
```

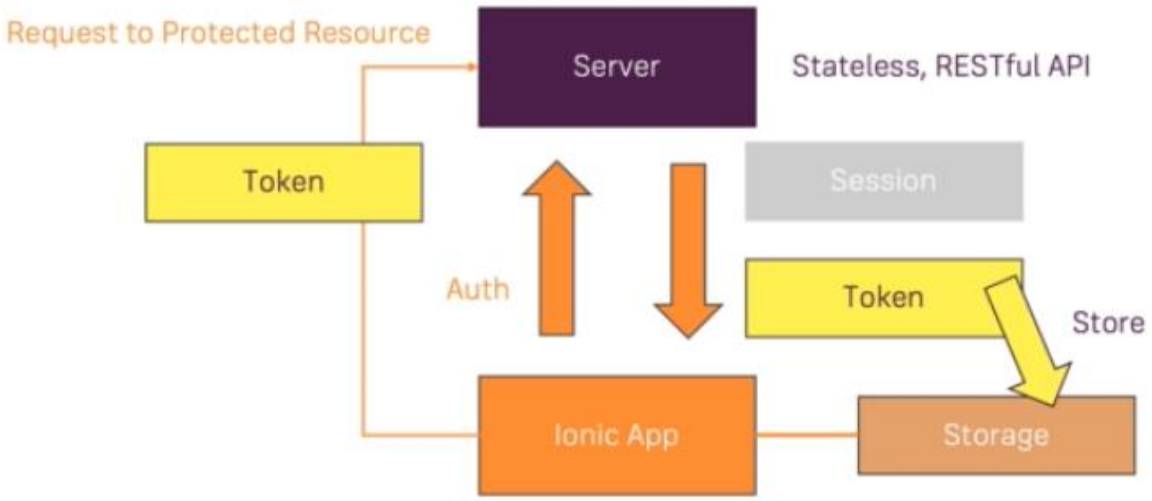
Slika 15 - ts datoteka kompoenente birača slike

Na slikama 13. i 14. vidi se kako se referenca veže na html element. Zatim se u metodi „onFileChosen()“ učitava slika pomoću „FileReader-a“ i poziva događaj sa kojim se slika šalje u komponentu izrade nove ponude, gdje se pokupi i šalje u obrazac. Nadalje, potrebno je još dodati opciju za direktno slikanje kod desktop računala (ili bilo kojih uređaja koji imaju kameru

a nisu android uređaji). Za to potrebano je samo uvesti pwa (Proegressive Web App) elemente na način kako je navedeno u dokumentaciji od Capacitora. Zadnja stvar koja nas čeka jest prebacivanje dobivene slike na server. U datoteci „places.service.ts“ definirati ćemo metodu „**uploadImage()**“ čiji će argument biti slika u datotečnom obliku. Zatim, pomoću http funkcije „post()“ ćemo poslati sliku na server tako da kao argumente navedemo url koji nam je Firebase generirao i podate koje želimo poslati. Naknadno ćemo metodu „uploadImage()“ korisiti u ts datoteci komponente nove ponude, gdje pri dohvaćanju podataka iz obrasca dohvatimo sliku i prenesemo je na server.

3.7 Autentifikacija korisnika

Autentifikaciju nećemo implementirati pomoću „sessiona“ već pomoću tokena jer radimo sa „RESTful API“ serverom. Dakle, autentifikacija će funkcionirati na način da pošeljemo autentifikacijske informacije na server i ukoliko on odobri te informacije nama će vratiti token. Taj token ćemo mi spremiti u memoriju aplikacije i zatim, kada budemo pristupali nekim resursima na serveru, slat ćemo zahtjev s tokenom kojeg će onda server, ukoliko je valjan, odobriti i poslati nam tražene resurse. Sav aplikacijski kod za autentifikaciju nalazit će se u „auth“ komponenti (/app/auth).



Slika 16 - Funkcioniranje autentifikacije pomoću servera

3.7.1 Registracija i prijava

Posebno za prijavu i registraciju koristiti ćemo različite URL-ove koje nalazimo u dokumentaciji od Firebasea. Njima ćemo dodati „API“ key koji nam također generira Firebase. Taj ključ ćemo spremiti u environment datoteku tako da ga svugdje možemo jednostavno

koristiti.Za registraciju, u service.ts datoteci definirati ćemo metodu „**signup()**“ kojoj ćemo proslijediti e-mail i lozinku. Napomena da se e-mail i lozinka ne trebaju validirati ovdje budući da su se već validirali u page.ts datoteci. Zatim ćemo u toj metodi pozvati http funkciju „post()“ kojoj ćemo kao argumente proslijediti:

1. URL – koji nam je generirao Firebase + „API ključ
2. Email, lozinka, „returnSecureToken = true“ (jer želimo da nam vrati token)

Također ćemo definirato „**login()**“ metodu koja će u bazi biti ista kao i „signup()“ metoda, jedina razlika će biti u URL-u po kojem će server raspoznati o kojoj metodi se radi.

Nakon toga za podatke koji će nam biti vraćeni stvorit ćemo interface AuthResponseData, a on će sadržavati:

- kind: string – tip requesta poslan na server
- idToken: string – id oznaka tokena
- email: string – e-poštu novo stvorenog korisnika
- refreshToken: string – dozvoljava da osvježimo token koji će isteći, budući da na Firebase svaki token traje jedan sat
- localId: string – id oznaka korisnika
- expiresIn: string – oznaka kada će token isteći
- registred: boolean – je li e- pošta prijašnje bio validirana

U page.ts datoteci definirati ćemo metodu „**authenticate()**“ u kojoj ćemo ovisno radi li se o prijavi ili registraciji pozivati odgovarujuću metodu iz service.ts datoteke u kojoj nam se nalaze „**signup()**“ i „**login()**“ metode. Što se tiče upravljanja pogreškama pri registraciji ili prijavi, u page.ts datoteci definirati ćemo metodu „**showAlert()**“ u kojoj ćemo implementirati prozorčić sa naslovom i određenom porukom koji se pojavi korisniku na ekranu ukoliko je napravio pogrešku u prijavi/registraciji. Zatim. u „authenticate()“ metodi pri radu za observable imamo određen dio gdje upisujemo kod ukoliko se pojavila neka greška. Server samostalno određuje i vraća tip pogreške kojem onda pristupamo i ovisno o tome koji je tip pogreške šaljemo prilagođenu poruku „showAlert()“ metodi koja će je onda prikazati korisniku.

3.7.2 Upravljanje tokenom

Dakle, kao što je prijašnje navedeno, kada token dobijemo od servera trebamo ga spremiti u lokalnu memoriju. Prvo ćemo kreirati „user.model.ts“ datoteku u kojoj ćemo stvariti model korisnika. Sadržavati će:

1. id
2. email
3. token
4. token istječe

U toj datoteci ćem, također, definirati metodu pomoću koje ćemo moći pristupati tokenu korisnika. Zatim, u service.ts datoteci definirati ćemo metodu **„setUserData()“**, kojoj ćemo poslati sve podatke o koriisniku koje smo dobili kod prijave/registracije i stvoriti objekt korisnika koji ćemo onda spremiti u memoriju s „BehaviourSubject“. Još je potrebno definirati dvije metode:

- userIsAuthenticated() – ona će nam vraćati boolean vrijednost a pristupajući tokenu i provjerom njegove valjanosti ustanoviti će ako je korisnik autentificiran
- userId() – pristupajući id korisnika ako postoji, vratiti će njegovu id oznaku

Trenutno ako korisnik osvježi stranicu u aplikaciji izgubiti će sve podatke i biti će usmjeren natrag na autentifikaciju iz razloga što sve te podatke čuvamo samo lokalno. Unatoč tomu, možemo korisiti „Capacitor storage“ kako bismo pospremili podatke: id korisnika, email, token. To ćemo implementirati u metodi **„storeAuthData()“** u service.ts“ datoteci, koristiti ćemo funkciju „set()“ kojoj čemo proslijediti proizvoljan ključ i podatke koje želimo spremiti u spremište. Zatim ćemo definirati metodu „**autologin()**“ u kojoj ćemo pristupati tim podacima koje smo spremali, na način da kao argument funkciji „get()“ proslijedimo ključ koji smo prethodno definirali. Nakon toga, dobivene ćemo podatke spremiti kao javascript objekt i kreirati novog korisnika pomoću tih podataka te zatim dodati novog korisnika sa „next()“ budući da je atribut u kojem spremamo korisnike tipa „Behaviour Subject“. Nadalje, „autologin()“ metoda vraćati boolean vrijednost ovisno o tome je li korisnik uspješno logiran ili nije. Zatim, datoteka „auth.guard.ts“ koja nam služi kao čuvar da korisnici ne smiju pristupati stranicama aplikacije ukoliko nisu prijavljeni; ona će pozivati metodu „autologin()“ te ukoliko i ta metoda vraća false vrijednost tek onda će preusmjeriti korisnika na autentifikaciju. U suprotnom će izvršiti metodu „autologin()“. Pomoću metode **„logout()“** ćemo odjaviti korisnika kad je potrebno i obrisati sve podatke iz spremišta. Zatim ćemo u „service.ts“ datoteci definirati metodu **„autoLogout()“**, koja će se pobrinuti da korisnik bude odjavljen kada istekne token. Tako da ćemo toj metodi kao argument poslati trajanje, i kada to trajanje završi izvršiti će se metoda „logout()“. Zadnja stvar koju je potrebno napraviti je dodati token pri svakom slanju zahtjeva na server, i na server strane koda dodati provjeru postoji li token i ako ne postoji

onesposobiti pristup aplikaciji. Pri svakom slanju zahtjeva ćemo onda slati i token, i također ključnu riječ „Bearer“, koju ćemo provjeravati postoji li na server strani. Token ćemo slati pozivajuću getter koji imamo u service.ts datoteci.

3.8 Stiliziranje aplikacije

Za organiziranje prikaza podataka u svakoj html datoteci koristio se „<ion-grid>“, koji sadži retke, a svaki redak sadrži 12 stupaca. Na taj način podaci se mogu dodati u stupce, odrediti koliko stupaca će jedan stupac biti širok, te koliko će biti razmaka. Rezultat je jednostavno upravljanje podacima koji trebaju biti prikazani, i uredno dizajniran kostur stranice. Kod autorizacije, pri unosa E-maila ili lozinke, te pristiskom na jedan od navedenih, tekst odlazi iznad prostora za unos koji odmah proširuje kako bi korisnik imao veću preglednost teksta koji unosi. To se u Ionicu postiže koristeći unutar „<ion-item>“, kombinaciju „<ion-label>“ i „<ion-input“>. Label-u se dodaje argument „floating“ kako bi korisničkim pritiskom otputovao iznad i smanjio veličinu. Veće kartice koje aplikaciji predstavljaju jedan smještaj implementirane su pomoću „<ion-card>“. Tada se lako dodaju ostale elementi unutar kartice kao što su naslov, opis, slika itd.. . Ostale kartice sa smještajima koje su konkretno prikazane na stranici potraži smještaje realizirane su pomoću „<ion-item>“ koji se nalazi unutar „<ion-virtual-scroll>“. Na taj način se implementira „virtual scrolling“ sa elementima kojima se može dodati određena visina, naslov, slika i ostalo. „Virtual scrolling“ funkcionira na način da su u „prozoru“ koji korisnik trenutno vidi svi ti elementi učitani i renderirani, a elementi koje korisnik ne vidi odnosno oni koji se nalaze malo iznad ili ispod korisničkog vidnog prozora su učitani, ali ne renderirani, dok oni elementi koji se nalaze daleko od korisničkog vidnog prozora nisu niti učitani niti renderirani. Na taj način smanjuje se memorijski teret i aplikacija ne mora svaki put učitavati sve elemente koji postoje na toj stranici. Pri pregledavanju svih korisničkih ponuda ili rezervacija postoji opcija brisanja ili uređivanje do koje se dolazi klizanjem elementa s desna na lijevo. To je implementirano pomoći „<ion-item-sliding>“ unutar kojeg se dodaje „<ion-item-option>“. Zatim se unutar te opcije može dodati slušač klika koji onda izvršava određenu funkciju. Unutar korijenske html datotke kreiran je izbornik, a ikone su mu dodane sa Ionica. Taj izbornik se naknadno dodaje u bilo koju stranicu aplikacije pomoću „<ion-menu-button>“ gdje se onda prikaže kao „hamburger icon“ na koji se klikom otvara potpuni prethodno stvoreni izbornik. Unutar komponente mjesta dodane su kartice sa mijenjanje između ponude i potražnje smještaja. To je implementirano pomoću „<ion-tabs>“, u kojima postoje opcije kao što odabir lokacije kartica na stranici i ostalo. Što se tiče boja, u „/theme/variables.scss“ datoteci definire su sve boje koje se koriste u aplikaciji i njihov kontrast kako bi odgovarale svakoj

pozadini. U „Ionic“ dokumentaciji postoji generator boja u kojemu se odredi nekoliko glavnih boja koje developer namjerava koristiti u aplikaciji. Zatim se automatski generira kod, koji se može kopirati i zalijepiti u prethodno navedeno datoteku, gdje svaka boja ima svoje ime pomoću kojeg joj se pristupa. Nadalje, u svakom dijelu aplikacije možemo pristupiti bilo kojoj boji koristeći njezino ime. Ionic samostalno kroz aplikaciju mijenja ton te boje kako bi se bolje uklopila u stil stranice.

4. Zaključak

Iako „Ionic-Angular“ pristup razvoja hibridnih web aplikacija nudi jednostavan i visoko funkcionalan koncept, moram priznati, kao osoba koja se prvi put srela sa svim tim novim alatima, kako cijeli pristup nije bilo toliko jednostavan. Neki od ostalih alata slični „Ionic-u“ su: „Flutter-Dart“ platforma, „React Native“ i „NativeScript“. Te platforme većinom se baziraju na pisanju programskog koda koji kasnije bude kompajliran u izvorni kod koji se izvršava na targetiranom uređaju. S druge strane, „Ionic“ se bazira na razvoju web aplikacije koja će biti omotana u pravu izvornu aplikaciju. „Ionic“ je time dodatno uključen u izradu „Capacitor“ projekta, što je i razlog zašto sam odabrao „Capacitor“ umjesto slične platforme kao što je „Cordova“. Što se tiče iskorištenosti koda, „Ionic“ nadmašuje svu konkurenciju, a i pri jednostavnosti izrade korisničkog sučelja nalazi se na vrhu, dok „ReactNative“ dosta podbacuje. Glede popularnosti i perfomansa „Ionic“ se nalazi na zadnjim mjestima, a vodeći su „ReactNative“ i „Flutter“, no to se ne bi trebalo previše uzimati u u obzir budući da su razine u performansama sitne. „Ionic“ se umjesto s „Angularom“ može koristiti i s platformama kao što su „React“ i „Vue“. Postoji bogata dokumentacija kao i mnogo izvora na webu koji potvrđuju mišljenje toga mislim da je „Ionic-Angular“ pristup jedan od najefikasnijih.

5. Literatura

1. Haverbeke, Marijn, Eloquent Javascript third edition, No Starch Press, San Francisco, 2019
2. Lim, Greg, Beginning Angular 2 With Typescript, CreateSpace, Scottsdale, 2017

6. Internetski izvori

1. https://ionicframework.com, 15.8.2019, 15:50
2. https://angular.io, 15.8.2019, 13:30
3. https://firebase.google.com, 15.8.2019, 14:50
4. https://searchsoftwarequality.techtarget.com, 25.8.2019, 21:20
5. https://capacitor.ionicframework.com, 25.8.2019, 21:30
6. https://stackoverflow.com/, 15.8.2019, 13:45
7. https://www.youtube.com, 15.8.2019, 14:00